LOVELY PROFESSIONAL UNIVERSITY

CSE 316 - OPERATING SYSTEMS

Question No 02

REPORT FILE ON QUESTION NO 02

SUBMITTED BY :
Ganeshwar Sahoo - RK21GP61

Guided by:
Mrs Cherry Khosla Mam

Table of Contents

Your paragraph text

Question

Consider a scheduling approach which is non pre-emptive similar to shortest job next in nature. The priority of each job is dependent on its estimated run time, and also the amount of time it has spent waiting. Jobs gain higher priority the longer they wait, which prevents indefinite postponement. The jobs that have spent a long time waiting compete against those estimated to have short run times.

The priority can be computed as : Priority = 1+ Waiting time / Estimated run time Write a program to implement such an algorithm.

Ensure:
1. The input is given dynamically at run time by the user .
2. The priority of each process is visible after each unit of time .
3. The gantt chart is shown as an output.
4. Calculate individual waiting time and average waiting time.

Introduction

Scheduling algorithms are an important aspect of operating systems. In non-preemptive scheduling, the CPU remains allocated to a process until the process completes its execution. The shortest job next (SJN) scheduling algorithm prioritizes processes based on their expected running time, completing short processes first. However, SJN may lead to indefinite postponement of long processes, leading to lower throughput.

To mitigate the issues with SJN, we can adopt an approach where the priority of each process is dependent on its estimated run time and the time it has spent waiting. Jobs gain higher priority the longer they wait, which prevents indefinite postponement.

Scheduling algorithms play a crucial role in the efficient use of resources in operating systems. Non-preemptive scheduling algorithms allow a process to hold the CPU until it completes its execution or blocks on a resource. In such cases, it becomes crucial to prioritize the processes based on some criteria to ensure that the CPU is used optimally.

The shortest job next (SJN) scheduling algorithm is a non-preemptive scheduling algorithm that prioritizes processes based on their expected running time. This algorithm is effective in minimizing the average waiting time but may lead to indefinite postponement of long processes, resulting in lower throughput.

The scheduling approach that we have discussed here is similar to SJN but takes into account the waiting time of a process in addition to its estimated run time. This approach ensures that processes that have been waiting for a long time are prioritized higher, preventing indefinite postponement. The priority of each process is calculated based on the formula:

Brief Description

This problem is about implementing a scheduling algorithm that prioritizes jobs based on their estimated run time and how long they have been waiting in the queue. The algorithm is similar to shortest job next (SJN) but takes into account the waiting time of each job to prevent indefinite postponement of long-waiting jobs.

The priority of a job is calculated as follows:

Priority = 1 + Waiting time / Estimated run time

The program takes input dynamically from the user at run time. For each job, the user inputs the arrival time and estimated run time. The program then stores each job in a data structure (in this case, a priority queue) and executes them in order of priority.

At each unit of time, the program selects the job with the highest priority (i.e., the job with the longest waiting time and shortest estimated run time) and executes it. The waiting time and priority of the other jobs in the queue that are waiting are updated.

The scheduling algorithm described in this problem is known as Shortest Job First with Waiting Time (SJF-WT) or Shortest Job First with Aging. The basic idea is to prioritize jobs that require less time to complete, but also take into account how long a job has been waiting in the queue. This prevents long jobs from being indefinitely postponed in a queue with shorter jobs. The aging factor ensures that a job's priority increases with the passage of time it spends waiting in the queue.

Priority = 1 + Waiting time / Estimated run time

where waiting time is the amount of time the job has spent in the queue waiting to be executed, and estimated run time is the amount of time the job is expected to take to complete.

The program takes input dynamically from the user, which means that the user inputs the number of jobs to be scheduled, their arrival time, and their estimated run time during the program's execution. The program then stores each job in a data structure, such as a priority queue, and executes them based on their priority.

Logic of the Code

The program prompts the user to enter the number of jobs, and then for each job, the arrival time and run time are entered.

Each job is stored in a Job struct with an initial waiting time of 0 and priority of 1.

The Job struct has a ComparePriority class that defines the comparison operator for the priority queue. This ensures that the jobs are always ordered based on their priority.

The program uses a priority queue to store the jobs. At each unit of time, the program selects the job with the highest priority (i.e., the job with the longest waiting time and shortest estimated run time) and executes it.

The waiting time and priority of the other jobs in the queue that are waiting are updated.

After all the jobs have been executed, the program calculates the total waiting time and average waiting time for the jobs.

The program outputs the gantt chart that shows the order in which the jobs were executed, along with the priority of each job at the time of execution.

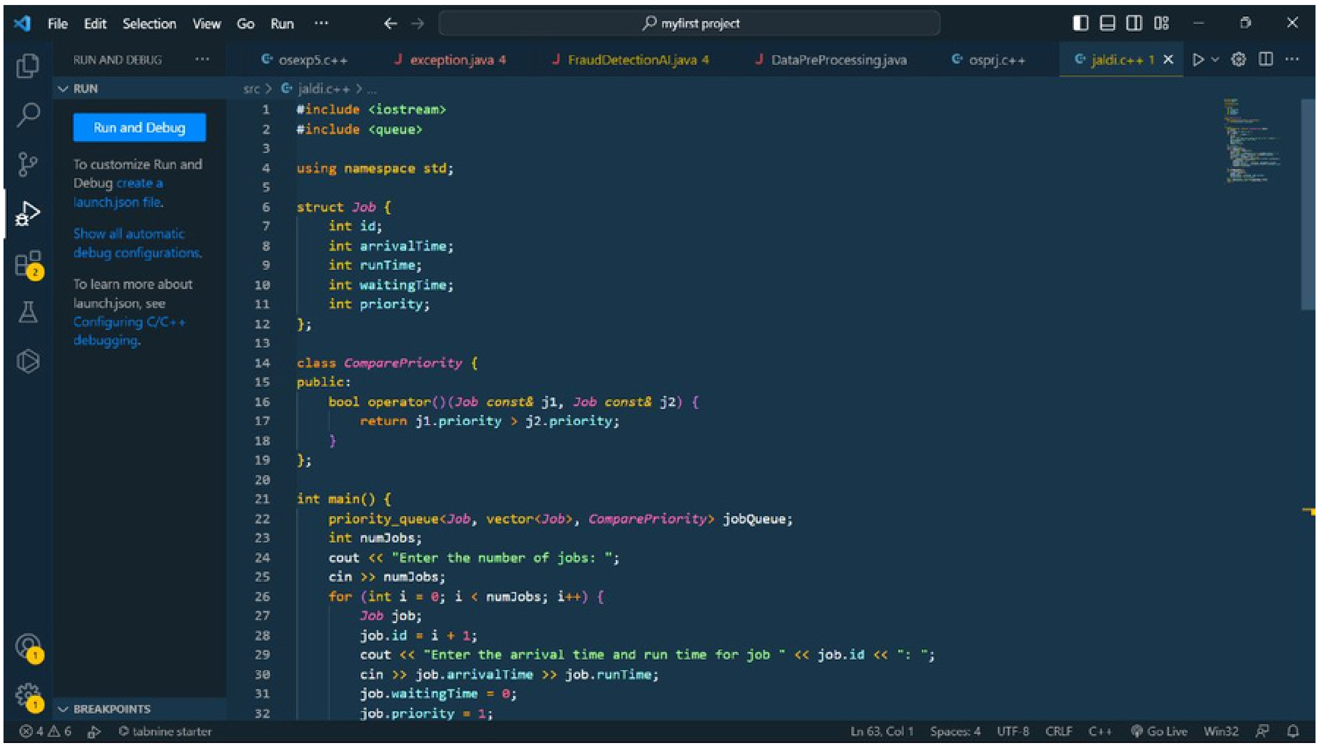
The program then outputs the average waiting time for the jobs.

Code

```
#include <iostream>
#include <queue>

using namespace std;

struct Job {
   int id;
   int arrivalTime;
   int runTime;
   int waitingTime;
   int priority;
};

class ComparePriority {
public:
   bool operator()(Job const& j1, Job const& j2) {
      return j1.priority > j2.priority;
   }
};

int main() {
   priority_queue<Job, vector<Job>, ComparePriority> jobQueue;
   int numJobs;
   cout << "Enter the number of jobs: ";
   cin >> numJobs;
   for (int i = 0; i < numJobs; i++) {
      Job job;
      job.id = i + 1;
      cout << "Enter the arrival time and run time for job " << job.id << ": ";
      cin >> job.arrivalTime >> job.runTime;
      job.waitingTime = 0;
      job.priority = 1;
      jobQueue.push(job);
   }

}
```

```
 int currentTime = 0;
  cout << "Gantt chart:\n";
  while (!jobQueue.empty()) {
    Job currentJob = jobQueue.top();
    jobQueue.pop();
    cout << "Job " << currentJob.id << " (" << currentJob.priority <<
") ";
    currentJob.waitingTime += currentTime -
currentJob.arrivalTime;
    currentTime += currentJob.runTime;
    cout << "[" << currentTime << "]" << endl;
    while (!jobQueue.empty() && jobQueue.top().arrivalTime <=
currentTime) {
      Job waitingJob = jobQueue.top();
      jobQueue.pop();
      waitingJob.waitingTime += currentTime -
waitingJob.arrivalTime;
      waitingJob.priority = 1 + waitingJob.waitingTime /
waitingJob.runTime;
      jobQueue.push(waitingJob);
    }
  }
  int totalWaitingTime = 0;
  while (!jobQueue.empty()) {
    Job job = jobQueue.top();
    jobQueue.pop();
    job.waitingTime += currentTime - job.arrivalTime;
    totalWaitingTime += job.waitingTime;
  }
  double avgWaitingTime = (double)totalWaitingTime / numJobs;
  cout << "Average waiting time: " << avgWaitingTime << endl;
  return 0;
```

SCREENSHOTS

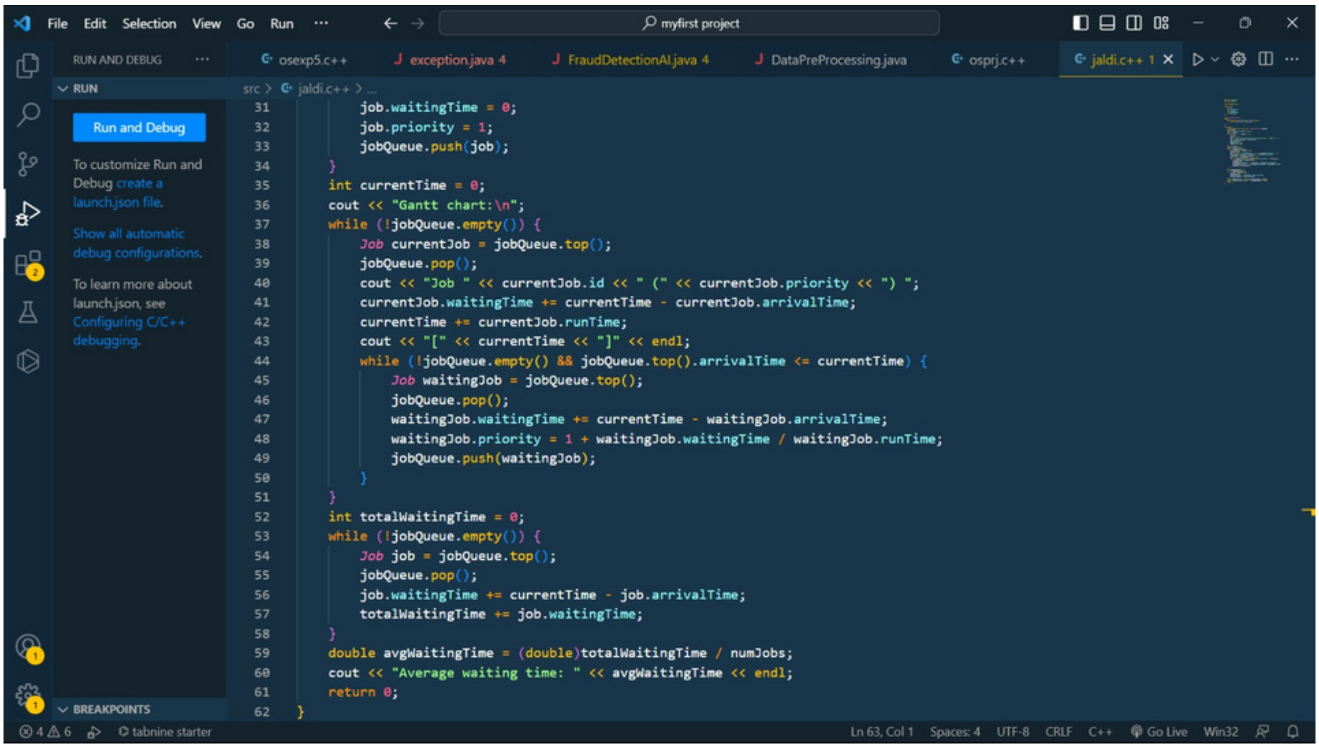


To customize Run and Debug create a launch.json file.

Show all automatic debug configurations.

To learn more about launch.json, see Configuring C/C++ debugging.

```cpp
#include <iostream>
#include <queue>

using namespace std;

struct Job {
    int id;
    int arrivalTime;
    int runTime;
    int waitingTime;
    int priority;
};

class ComparePriority {
public:
    bool operator()(Job const& j1, Job const& j2) {
        return j1.priority > j2.priority;
    }
};

int main() {
    priority_queue<Job, vector<Job>, ComparePriority> jobQueue;
    int numJobs;
    cout << "Enter the number of jobs: ";
    cin >> numJobs;
    for (int i = 0; i < numJobs; i++) {
        Job job;
        job.id = i + 1;
        cout << "Enter the arrival time and run time for job " << job.id << ": ";
        cin >> job.arrivalTime >> job.runTime;
        job.waitingTime = 0;
        job.priority = 1;
```

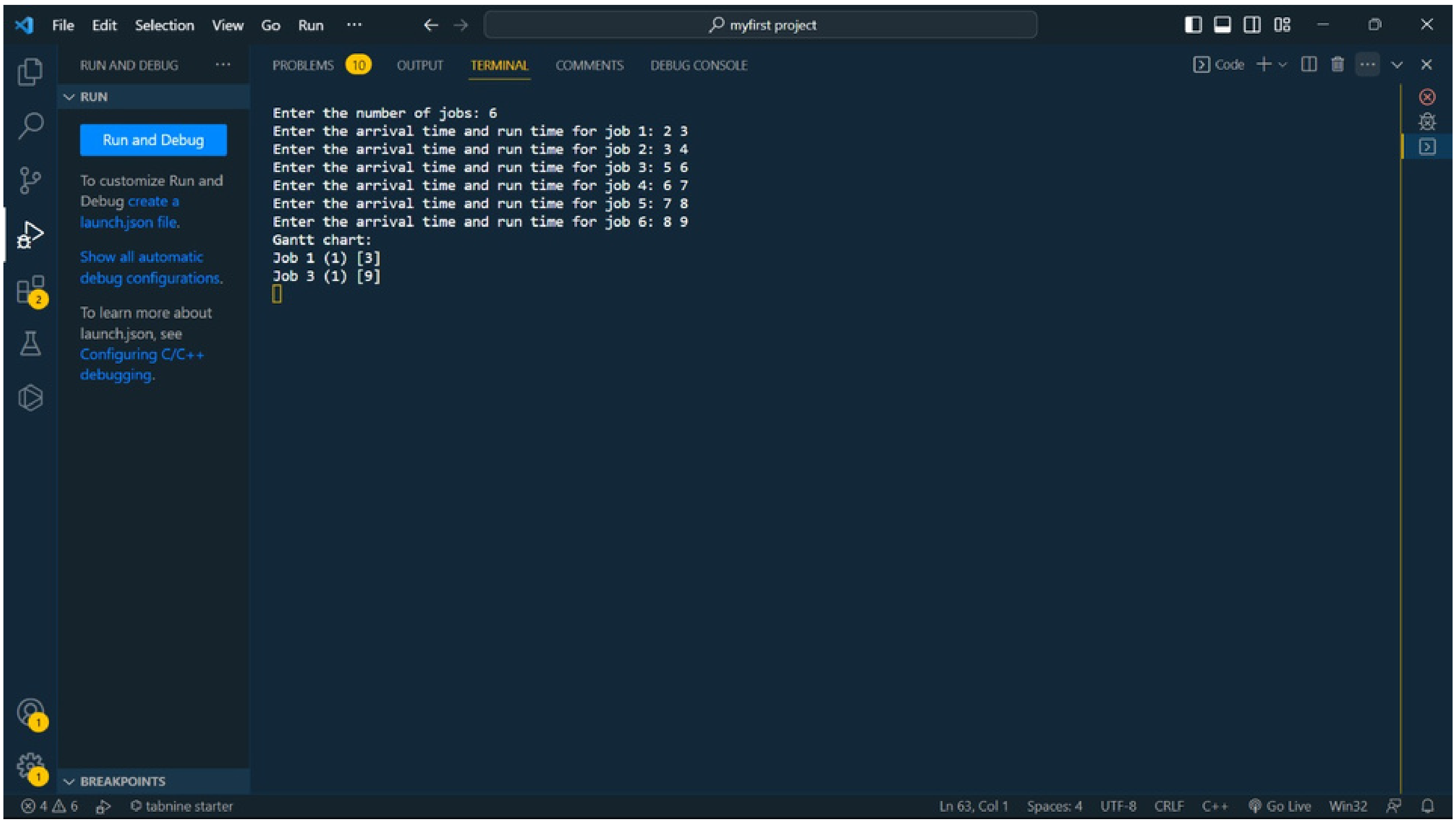


To customize Run and Debug create a launch.json file.

Show all automatic debug configurations.

To learn more about launch.json, see Configuring C/C++ debugging.

```cpp
        job.waitingTime = 0;
        job.priority = 1;
        jobQueue.push(job);
    }
    int currentTime = 0;
    cout << "Gantt chart:\n";
    while (!jobQueue.empty()) {
        Job currentJob = jobQueue.top();
        jobQueue.pop();
        cout << "Job " << currentJob.id << " (" << currentJob.priority << ") ";
        currentJob.waitingTime += currentTime - currentJob.arrivalTime;
        currentTime += currentJob.runTime;
        cout << "[" << currentTime << "]" << endl;
        while (!jobQueue.empty() && jobQueue.top().arrivalTime <= currentTime) {
            Job waitingJob = jobQueue.top();
            jobQueue.pop();
            waitingJob.waitingTime += currentTime - waitingJob.arrivalTime;
            waitingJob.priority = 1 + waitingJob.waitingTime / waitingJob.runTime;
            jobQueue.push(waitingJob);
        }
    }
    int totalWaitingTime = 0;
    while (!jobQueue.empty()) {
        Job job = jobQueue.top();
        jobQueue.pop();
        job.waitingTime += currentTime - job.arrivalTime;
        totalWaitingTime += job.waitingTime;
    }
    double avgWaitingTime = (double)totalWaitingTime / numJobs;
    cout << "Average waiting time: " << avgWaitingTime << endl;
    return 0;
}
```



```
Enter the number of jobs: 6
Enter the arrival time and run time for job 1: 2 3
Enter the arrival time and run time for job 2: 3 4
Enter the arrival time and run time for job 3: 5 6
Enter the arrival time and run time for job 4: 6 7
Enter the arrival time and run time for job 5: 7 8
Enter the arrival time and run time for job 6: 8 9
Gantt chart:
Job 1 (1) [3]
Job 3 (1) [9]
```